2010

Bit-rate Aware Reconfigurable Architecture For H.264/avc Deblocking Filter

Rakan Khraisha
University of Central Florida

BIT-RATE AWARE RECONFIGURABLE ARCHITECTURE FOR H.264/AVC DEBLOCKING FILTER

by

RAKAN KHRAISHA
B.S. University of Jordan, 2007

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2010

ABSTARCT

In H.264/AVC, DeBlocking Filter (DBF) achieves bit rate savings and it is used to improve visual quality by reducing the presence of blocking artifacts. However, these advantages come at the expense of increasing computational complexity of the DBF due to highly adaptive mode decision and small 4x4 block size. The DBF easily accounts for one third of the computational complexity of the decoder. The computational complexity required for various target applications from mobile to high definition video applications varies significantly. Therefore, it becomes apparent to design efficient architecture to adapt to different requirements.

In this work, we exploit the scalability on both the hardware level and the algorithmic level to synergize the performance and to reduce computational complexity. First, we propose a modular DBF architecture which can be scaled to adapt to the required computing capability for various bit-rates, resolutions, and frame rates of video sequences. The scalable architecture is based on FPGA using dynamic partial reconfiguration. This desirable feature of FPGAs makes it possible for different hardware configurations to be implemented during run-time. The proposed design can be scaled to filter up to four different edges simultaneously, resulting in significant reduction of total processing time. Secondly, our experiments show by lowering the bit rate of video sequences, significant reduction in computational complexity can be achieved by the increased presence of skipped macroblocks, thus, avoiding redundant filtering operations. The implemented architecture has been evaluated using Xilinx Virtex-4 ML410 FPGA board. The design can operate at a maximum frequency of 103 MHz. The reconfiguration is done through Internal Configuration Access Port (ICAP) to achieve maximum performance needed by real time applications.

To my parents

ACKNOWLEDGMENTS

The author wishes to thank the following for their tremendous contribution to this work:

Dr. Jooheung lee as a mentor and advisor, for which this work would not have been possible without him.

Dr. Ronald Demara and Dr. Jun Wang for their encouragement as professors and committee members.

Dr. Issa Batarseh for his advising and support throughout this journey.

Muneer Masad for his friendship and expertise.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

1.1: H.264/AVC Overview

The H.264/AVC is the latest video coding standard developed by the cooperation between ITU-T and ISO/IEC standardization organizations in 2003 [1], [2].

Compared to its predecessors, such as H.261/3 and MPEG-1/2/4, it provides improved video compression efficiency. This is due to the combination of advanced video coding techniques, such as variable block-size motion estimation with quarter-pixel resolution, intra prediction in the spatial domain, integer 4x4 DCT transform, context adaptive variable length coding, and in-loop adaptive DeBlocking Filter (DBF).

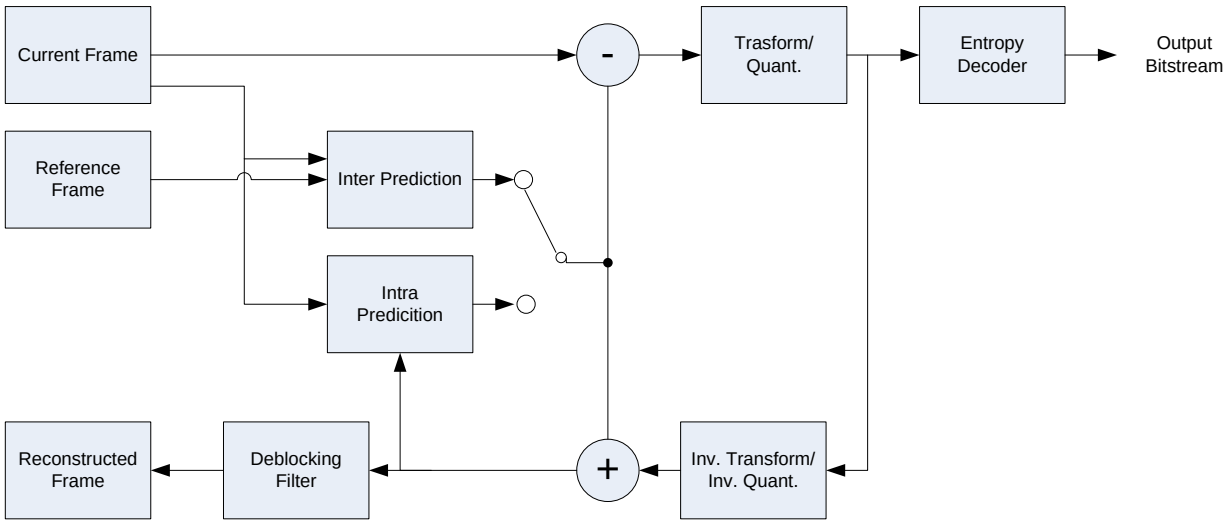

Figure 1: H.264 video encoder.

Figure 1 shows H.264 encoder. The input frame, which is divided into macroblocks, is encoded in either Inter or Intra prediction modes. In Intra mode the prediction is made from samples from the current frame that already have been encoded. In Inter mode, motion compensation prediction is used from reference frames. The current frame is subtracted from the predicted frame to produce a residual that is transformed, quantized, and entropy encoded. The encoder also decodes current frames, and DBF is applied for future prediction.

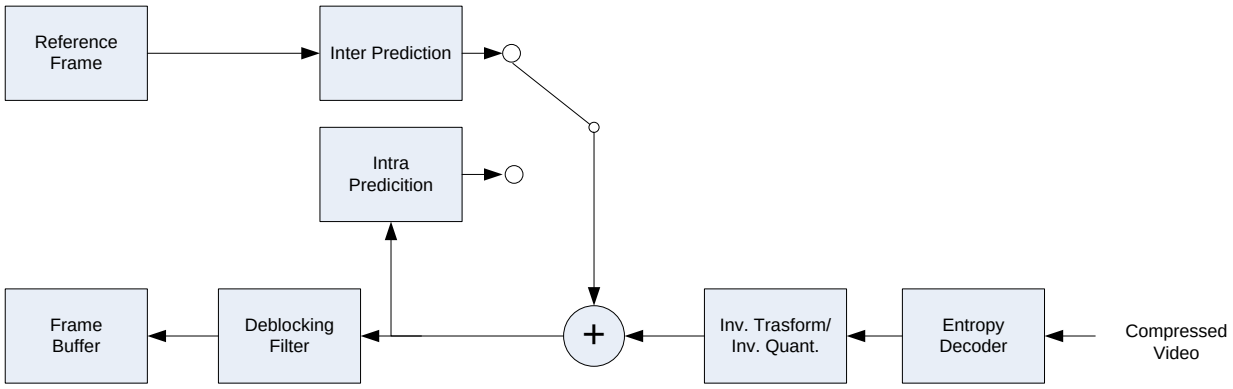

Figure 2: H.264 video decoder.

Figure 2 shows H.264 video decoder. The decoder receives the compressed video frames, which are reverse quantized and transformed, and then added to predicted blocks which is the same as the encoder. Finally the decoded frame is filtered and stored for later processing.

1.2: Inter Frame Prediction

Consecutive video frames being transmitted have similar data between them. Motion Estimation (ME) is used to remove temporal redundancies to achieve better compression efficiency. In Inter mode macroblocks which consists of 16x16 pixels are predicted from previously encoded video frames. The prediction is made using Motion Vectors (MV) which is the offset between the

predicted block and the location from the already encoded reference frame. In Figure 3 H.264/AVC supports various block sizes (from 16x16 to 4x4). This way better coding efficiency

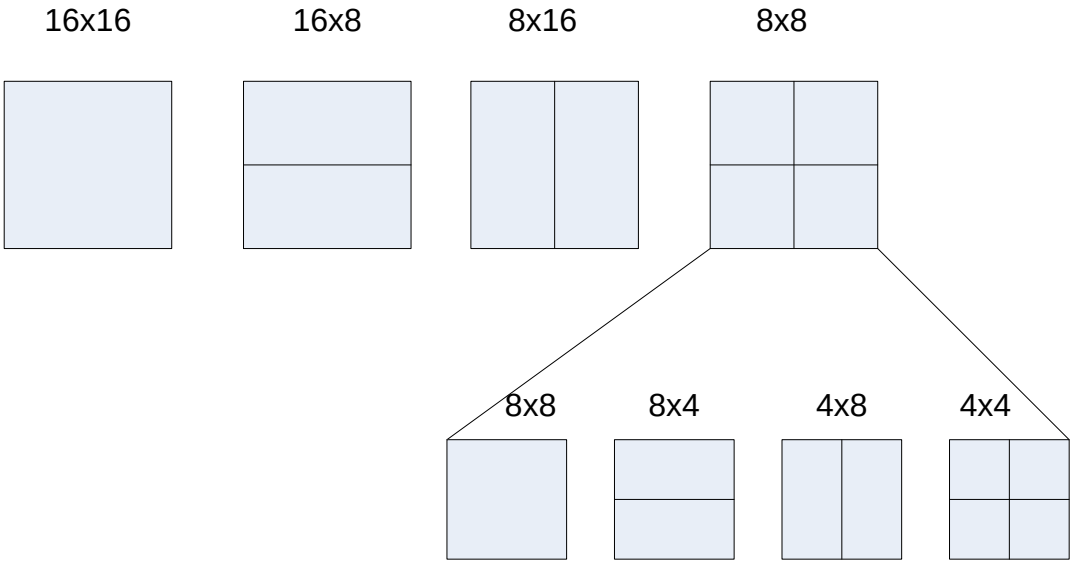

Figure 3: Macroblcoks partitioning

could be achieved by finding the best matching block in previous referenced frames. Also in H.264/AVC multiple frames could be used for motion compensation prediction which is shown in Figure 4.

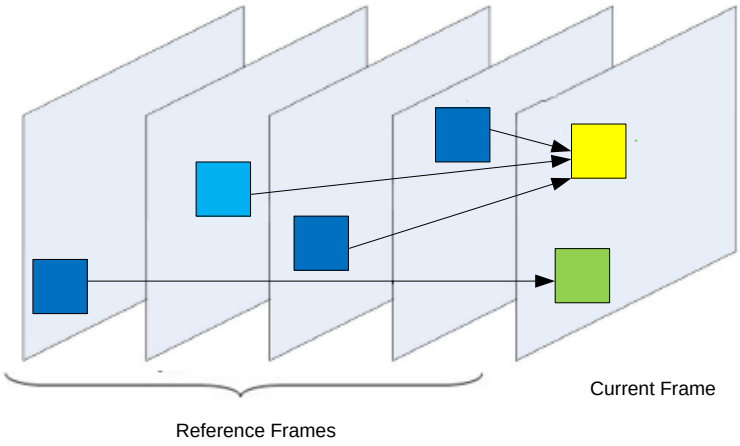

Figure 4: Multiple frames motion estimation

1.3: Intra Frame Prediction

In Intra prediction reference frames are not used. The prediction is made from previously encoded blocks from the same frame. There are two prediction modes; the first is Intra 4x4 prediction where there are nine prediction modes for each 4x4 subblock. The second one is intra 16x16 prediction where four prediction modes are available. Figure 5 shows possible eight prediction directions supported for Intra 4x4, the ninth mode is DC prediction mode where the mean is taken of all neighboring blocks of the top and the left of the current block.

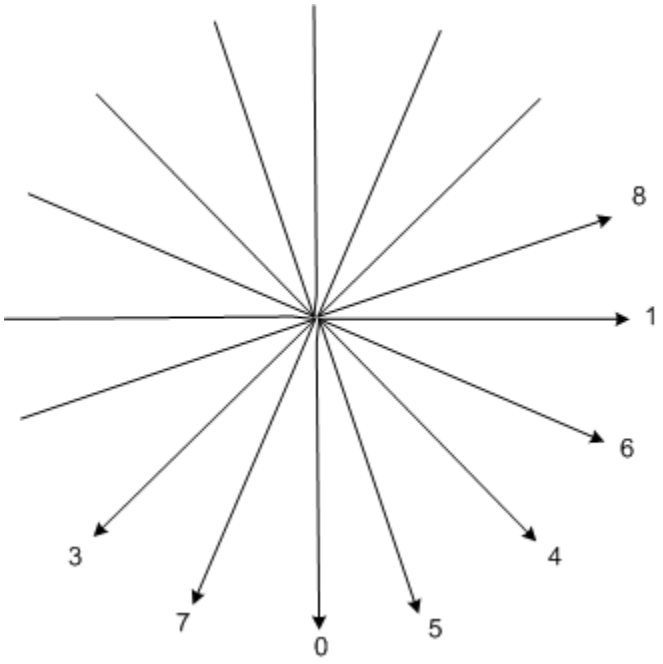

Figure 5: Prediction directions for Intra 4x4.

1.4: Transform and quantization

The transform stage is used to convert data from the image domain to the frequency domain. Macroblocks are transmitted according to the order shown in Figure 6. Depending on the data sent, one of three transforms is applied. A Hadamard transform is applied to macroblocks predicted in intra 16x16 mode and for 2x2 blocks of chroma DC coefficients. For all other 4x4 blocks a DCT based transform is applied.

Y

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 6 : Scanning order of blocks in H.264

The DCT applied is an integer transform which has the advantage of not losing coding accuracy, and it can be applied with shifts and additions which provides easier hardware implementation. The transform [3] is defined as:

$$a = \frac{1}{2}, \qquad b = \sqrt{\frac{5}{2}}$$

$$Y = C_f \mathbf{X} C_f^T \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

The quantization stage is used to provide compression by removing the high frequency components (which tends to be zero) created by the DCT transform. There are 52 quantization steps in the standard which provides more flexibility in the tradeoff between bit-rate and image quality. After the quantization stage data are reordered from low to high frequencies.

1.5: Deblocking Filter

The DBF is applied on edges of each 4x4 block in a Macroblock (MB), after inverse quantization and inverse transform. It improves the visual quality by reducing the presence of blocking artifacts in decoded video frames, which is caused by block-based transform, motion estimation, and quantization operations. Figure 7 shows the presence of blocky artifacts when encoded in H.264, and the improvement of visual quality when applying the deblocking filter.

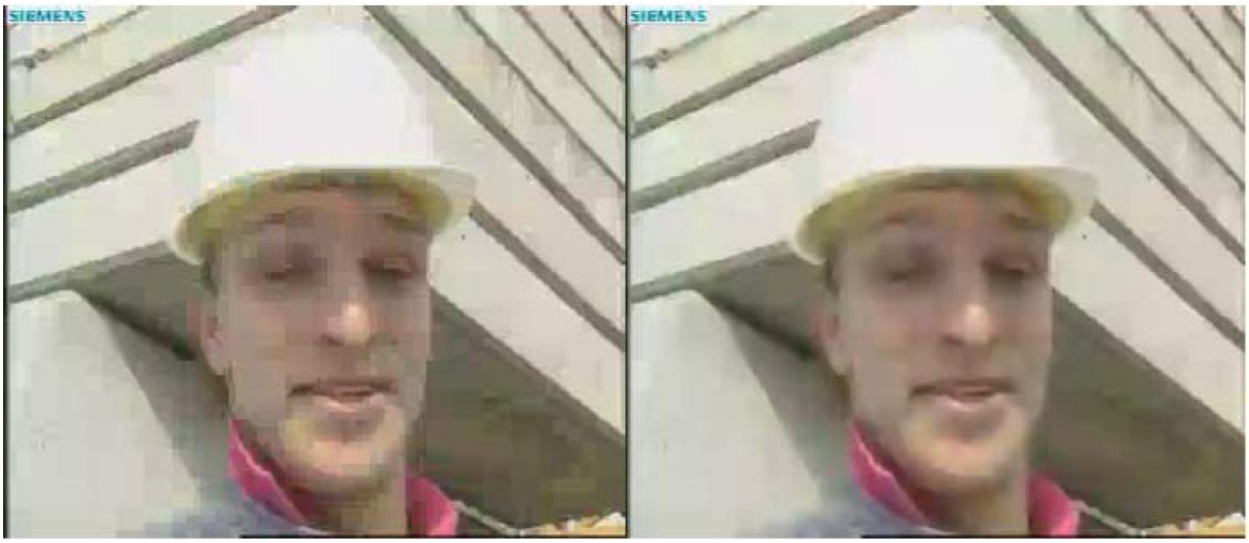

Figure 7 : Frame decoded without and with deblocking filter

1.5.1: Deblocking Filter Algorithm

H.264/AVC adaptive DBF algorithm reduces blocking artifacts created mainly by block-based transform and quantization operations. The filtering process consists of horizontal filtering across vertical edges and vertical filtering across horizontal edges. The restriction imposed by H.264/AVC on the filtering order is that horizontal filtering should precede vertical filtering, so all vertical edges are filtered before the horizontal ones. Figure 8 shows the filtering process, where an H.264/AVC filtering order is applied over edges in a macroblock. The DBF is highly adaptive 3~5 tap filter, it works on three different levels [4]: slice level, block level, and sample level. On the block level each edge is assigned a Boundary Strength (BS) value. The purpose of the BS value is to check if a blocking artifact may be present over an edge, and determine the strength of the filtering operation to be used on the edge. Table 1 shows the condition parameters for deciding the BS value depending on coded information.

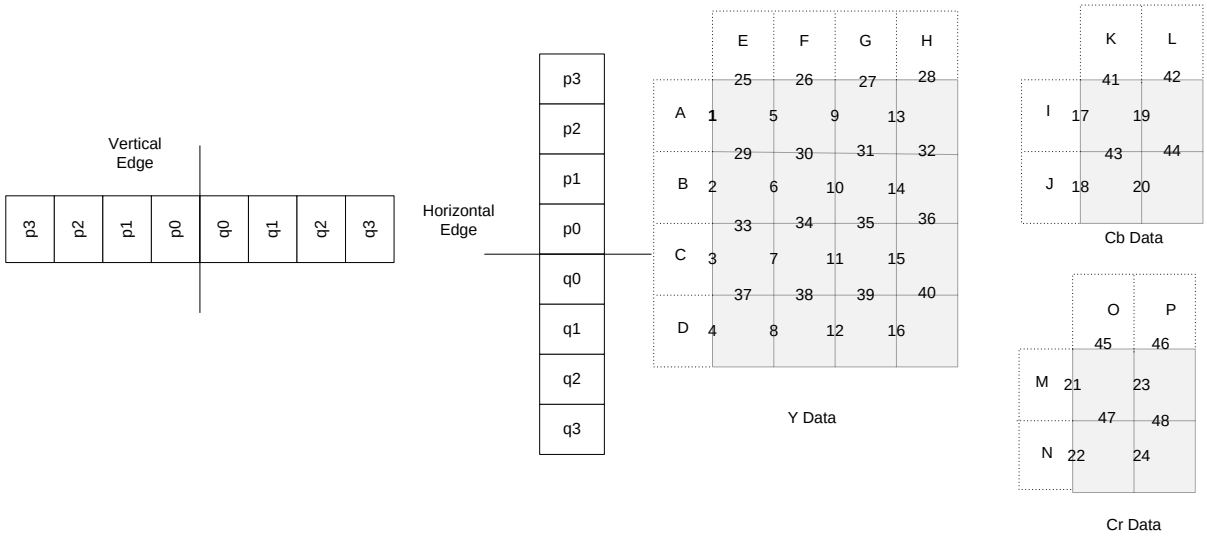

Figure 8: H.264/AVC filtering order of edges over a macroblock

Table 1: Determining of BS value

Conditions	BS value
Either of the blocks is intra coded and the edge is macro block boundary	BS=4
Either of the blocks is intra coded	BS=3
Either of the blocks contain coded coefficients	BS=2
The motion vector difference is >1	BS=1
else	BS=0

These parameters include the intra/inter mode prediction, the presence of non-zero residual coded coefficients, and the difference of motion vectors across the boundary. BS takes the value from 0 to 4. 0 stands for no filtering and 4 indicates maximum filtering. When BS is 4 a 3~5 tap

filter is applied depending on certain conditions, on the other hand when BS is between 1 and 3 a 4 tap filter is applied. On the sample level in addition to BS>0, three other conditions must hold so an actual edge should be filtered or not: $|p0 - q0| < \alpha, |p1 - p0| < \beta,$ and $|q1 - q0| < \beta$. The variables α and β are defined in the standard and increase with the increasing of the quantization parameter. Table 2 and Table 3 shows filter implementation according to BS value.

Table 2: Filter implementation when 0< BS < 4

Equations		**Output**
$\|p_2 - p_0\| < \beta$ TRUE	$\|q_2 - q_0\| < \beta$ TRUE	$\dot{p}_0 = (p_0 + \Delta_0)$ $\dot{p}_1 = (p_1 + \Delta_{p1})$ $\dot{q}_0 = (q_0 - \Delta_0)$ $\dot{q}_1 = (q_1 + \Delta_{q1})$
	$\|q_2 - q_0\| < \beta$ FALSE	$\dot{p}_0 = (p_0 + \Delta_0)$ $\dot{p}_1 = (p_1 + \Delta_{p1})$ $\dot{q}_0 = (q_0 - \Delta_0)$
$\|p_2 - p_0\| < \beta$ FALSE	$\|q_2 - q_0\| < \beta$ TRUE	$\dot{p}_0 = (p_0 + \Delta_0)$ $\dot{q}_0 = (q_0 - \Delta_0)$ $\dot{q}_1 = (q_1 + \Delta_{q1})$
	$\|q_2 - q_0\| < \beta$ FALSE	$\dot{p}_0 = (p_0 + \Delta_0)$ $\dot{q}_0 = (q_0 - \Delta_0)$

$$\Delta_0 = Min(Max(-c_0, \Delta_{oi}), c_0)$$
$$\Delta_{oi} = (4(q_0 - p_0) + (p_1 - q_1) + 4) \gg 3$$
$$\Delta_{p1} = Min(Max(-c_1, \Delta_{p1i}), c_1)$$
$$\Delta_{p1i} = (p_2 + ((p_0 + q_0 + 1) \gg 1) - 2p_1) \gg 1$$
$$\Delta_{q1} = Min(Max(-c_1, \Delta_{q1i}), c_1)$$
$$\Delta_{q1i} = (q_2 + ((p_0 + q_0 + 1) \gg 1) - 2q_1) \gg 1$$

Table 3: Filter implementation when BS=4

Equations			**Result**
$\lvert p_2 - p_0\rvert < \beta$ TRUE	$\lvert q_2 - q_0\rvert < \beta$ TRUE	$\lvert p_0 - q_0\rvert < (\alpha \gg 2) + 2$ TRUE	$\dot{p}_0 = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3$ $\dot{p}_1 = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2$ $\dot{p}_2 = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3$ $\dot{q}_0 = (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) \gg 3$ $\dot{q}_1 = (q_2 + q_1 + q_0 + p_0 + 2) \gg 2$ $\dot{q}_2 = (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) \gg 3$
		$\lvert p_0 - q_0\rvert < (\alpha \gg 2) + 2$ FALSE	$\dot{p}_0 = (2p_1 + p_0 + q_1 + 2) \gg 2$ $\dot{q}_0 = (2q_1 + q_0 + p_1 + 2) \gg 2$
	$\lvert q_2 - q_0\rvert < \beta$ FALSE	$\lvert p_0 - q_0\rvert < (\alpha \gg 2) + 2$ TRUE	$\dot{p}_0 = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3$ $\dot{p}_1 = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2$ $\dot{p}_2 = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3$ $\dot{q}_0 = (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) \gg 3$
		$\lvert p_0 - q_0\rvert < (\alpha \gg 2) + 2$ FALSE	$\dot{p}_0 = (2p_1 + p_0 + q_1 + 2) \gg 2$ $\dot{q}_0 = (2q_1 + q_0 + p_1 + 2) \gg 2$
$\lvert p_2 - p_0\rvert < \beta$ FALSE	$\lvert q_2 - q_0\rvert < \beta$ TRUE	$\lvert p_0 - q_0\rvert < (\alpha \gg 2) + 2$ TRUE	$\dot{p}_0 = (2p_1 + p_0 + q_1 + 2) \gg 2$ $\dot{q}_0 = (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) \gg 3$ $\dot{q}_1 = (q_2 + q_1 + q_0 + p_0 + 2) \gg 2$ $\dot{q}_2 = (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) \gg 3$
		$\lvert p_0 - q_0\rvert < (\alpha \gg 2) + 2$ FALSE	$\dot{p}_0 = (2p_1 + p_0 + q_1 + 2) \gg 2$ $\dot{q}_0 = (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) \gg 3$
	$\lvert q_2 - q_0\rvert < \beta$ FALSE	$\dot{p}_0 = (2p_1 + p_0 + q_1 + 2) \gg 2$ $\dot{q}_0 = (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) \gg 3$	

On the slice level, the amount of filtering can be changed through encoder offset values. These values changes α and β, and thereby increase or decrease the level of filtering that takes place.

1.6: Dynamic Partial Reconfiguration

Field Programmable Gate Arrays (FPGA) are digital integrated circuits comprised of configurable logic blocks, which are connected through programmable interconnects. Nowadays FPGA contains million of gates and it is used in myriad of applications like embedded processing, DSPs, communication, and Reconfigurable Computing (RC). This due to the fact that FPGA have low nonrecurring engineering costs, easy design modification, and faster time to market.

Most FPGAs today are SRAM based, which provides the flexibility for the designer of being able to program it multiple times. On the other side, anti-fuse based FPGAs are one time programmable, which make it useful in design protection against theft. To implement a design on the FPGA, a configuration file is uploaded to program logic blocks, routing switches, and I/O interface. Partial Reconfiguration (PR) [5] of an FPGA is done when a partial bit stream is loaded to the FPGA to configure some parts with new functionality. Xilinx provides this attractive feature of FPGAs. There are two ways to generate partial bitstreams: difference based and modular based [6]. Difference based partial reconfiguration [7] is useful in making small changes to the design. The changes can be made by using Xilinx FPGA_EDITOR, and then a partial bit stream is generated (using Xilinx BitGen) which contains the difference between the new design and the old one. This method only works if the original configuration bitstream is available.

In Modular based partial reconfiguration, certain regions in the hardware fabric which are called Partial Reconfigurable Regions (PRRs) can be time multiplexed with multiple functions. Figure 9 show a layout design with two PRRs. The PRRs can be reconfigured to do different functionalities while the static region remains unchanged.

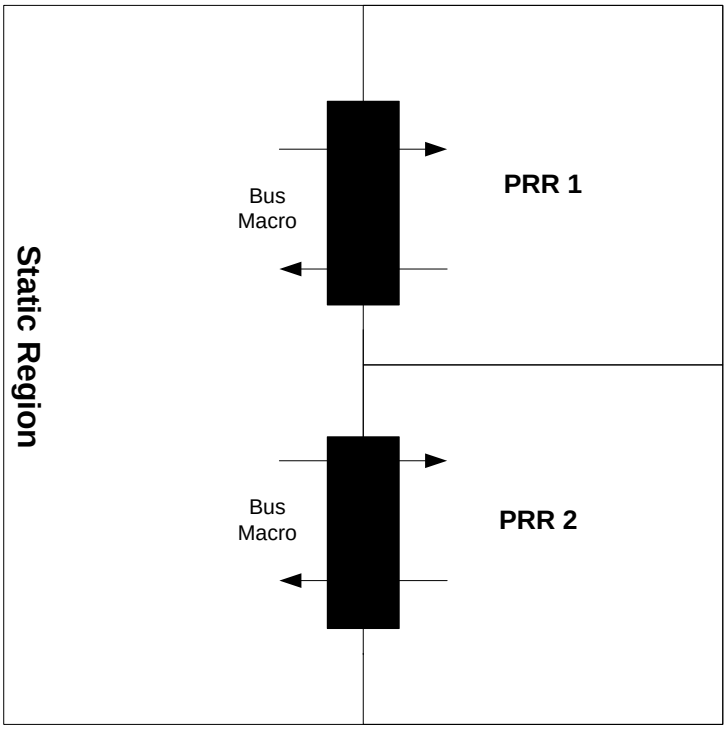

Figure 9: FPGA design layout with two PRRs

Bus Macros (BM) are used to communicate between the PRRs and static region. The main benefit of partial reconfiguration is that some parts of the design can be changed while the other parts remain unaffected. This implies faster reconfiguration time is achieved by not having to load the full bitstream into the FPGA, Less hardware area is used since different functions are time multiplexed, and reduced power consumption is achieved by loading blank bitstreams for regions that are not needed. Partial reconfiguration gives the designer the flexibility to make design changes during run-time without having to reconfigure the whole FPGA.

1.7: Motivation

First the DBF is the most computationally complex part of the H.264/AVC decoder. This due to highly adaptive mode decision and small 4x4 block size. The DBF can easily account for one third of the overall complexity of H.264 video decoder [4]. Secondly, the computational complexity required for various target applications from mobile to high definition video applications varies significantly. Therefore, it becomes important to design efficient architecture to adapt to different requirements, such as computing capability and power consumption in order to better utilize reconfigurable hardware resources.

Thirdly, previous works presented in the literature to build Intellectual Property (IP) cores in hardware have inherent limitations due to their fixed architectures with pre-determined computing capability, power consumption, and hardware area. Therefore, traditional IP cores targeting both ASIC and FPGA markets cannot change their architectures efficiently to adapt to changing environments. Our goal is to develop an IP core that is easily customizable to match the computing capability of the users' application and able to adapt itself to meet the varying workloads during run-time through dynamic partial reconfiguration. In this work, we combine the algorithmic and hardware scalability to design reconfigurable architecture of DBF engine by utilizing multiple reconfiguration regions which can be selectively used to support various bit-rates, resolutions, and frame rates of video sequences.

1.8: Thesis Organization

Chapter 2 investigates software and hardware implementations of DBF algorithm. Usually hardware implementations are preferred due to the computational complexity of the algorithm and also hardware implementations are more feasible to consumer products. Chapter 3 presents a modular reconfigurable DBF architecture, which supports various bit-rates, frame rates, and video resolutions. The chapter also discusses the implementation of the architecture using dynamic partial reconfiguration on FPGA. Chapter 4, is an extension to the previous chapter, where taking into the fact that by increasing compression the computational complexity can be reduced by the increased presence of skipped MBs, thus reducing the processing cycles needed for filtering operation. The chapter also discusses an algorithm where the number of processing elements can be changed dynamically with the value of quantization parameter.

CHATPTER TWO: SUPPORTING WORK

There have been some previous works related to the DBF algorithm, targeting for software and hardware implementations. In [8], parallel processing through SIMD (Single Instruction Multiple Data) is used to improve performance. In [9], they increase the degree of the instruction level parallelism of the dedicated processors to improve performance. Because of the high complexity of the deblocking filter, hardware solutions have been preferred.

2.1: Filter Order

In DBF algorithm, pixel values in a macroblock are read many times and intermediate pixel results are stored in temporary buffers for use in later stages. Changing filtering order improves data reuse and decrease memory bandwidth. In [10]-[17], one dimensional or two dimensional filtering orders are proposed to improve data reuse and reduce memory cycles required.

In [10] one dimensional filtering order is proposed to improve data reuse and reduce memory cycles. In [11] a semi two dimensional filtering order is employed where horizontal filtering is applied on the row and then vertical filtering is applied before moving to the next row in the macroblock. This way better data reuse is provided than one dimensional filtering and pixels can be written earlier to the main memory. In [12]-[14] better use of two dimensional filtering order is applied where horizontal filtering and vertical filtering alternate on the block, this way data reused is more efficient and local buffer size is reduced. In [15], two edge filters are applied and horizontal and vertical filtering occur simultaneously which greatly reduces filtering cycles for a macroblock.

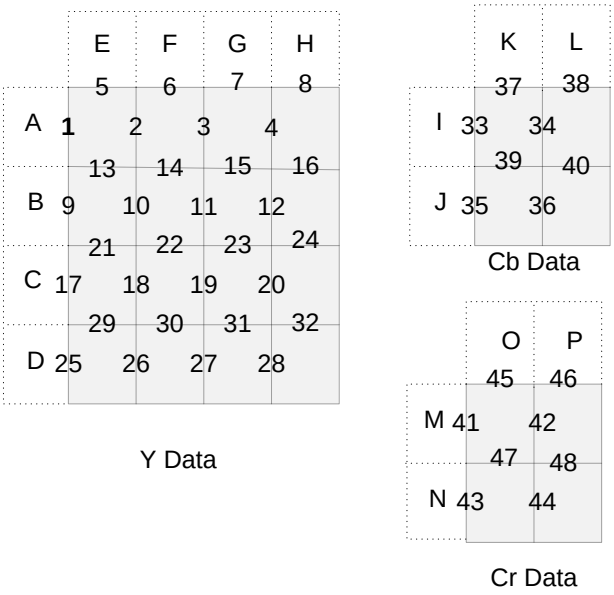

Figure 10: Filtering order in [7]

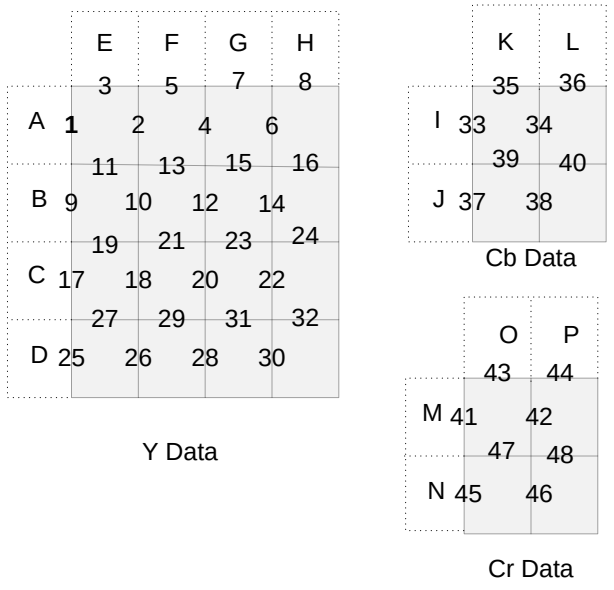

Figure 11: Filtering order in [8]-[10]

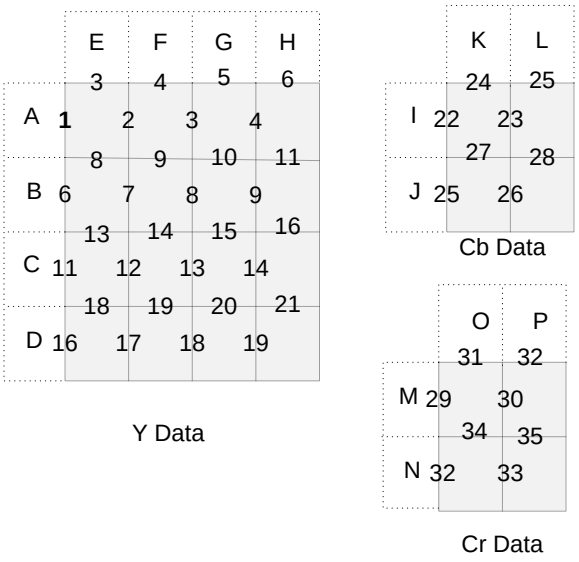

Figure 12: Filtering order in [11]

2.2: Filter Complexity

The DBF in H.264/AVC is the most complex part in the decoder [4]. Most works cited try to reduce DBF computational complexity. In [11], [15], and [17] the filter design is pipelined, this offer some advantages like reducing the critical path of the design, thereby increasing the clock speed of the design and reducing latency. However the filtering order is restricted to avoid data hazard. Other techniques have been used, in [18] multiple methods are used such as clock gating, glitch reduction techniques, and simplification of the datapath to reduce power consumption at the expense of performance. In [19], architecture is proposed to make use of spatial correlation of the pixels to reduce computational complexity.

2.3: Summary

DBF is the most computationally demanding module in H.264/AVC decoder. Software solutions have been suggested, but hardware solutions are usually preferred because they are more feasible

in consumer products. In this chapter a lot of works have been cited trying to reduce the complexity of the DBF. Some work tried to change filtering order to reuse data efficiently and to reduce memory bandwidth. Other works try to pipeline the design to reduce critical path and increase operating frequency. In others, simplification of the datapath and making use of spatial correlation of the pixels are used to reduce the computational complexity. However, previous works have inherent limitations due to their fixed architectures with pre-determined computing capability, power consumption, and hardware area. Therefore, traditional IP cores targeting both ASIC and FPGA markets cannot change their architectures efficiently to adapt to changing environments. Our goal is to develop an IP core that is easily customizable to match the computing capability of the users' application and able to adapt itself to meet the varying workloads during run-time through dynamic partial reconfiguration. In our work, we combine the algorithmic and hardware scalability to design reconfigurable architecture of DBF engine by utilizing multiple reconfiguration regions which can be selectively used to support various bit-rates, resolutions, and frame rates of video sequences.

CHAPTER THREE: SCALBALE H.264/AVC DEBLOCKING FILTER

3.1 Filter Order

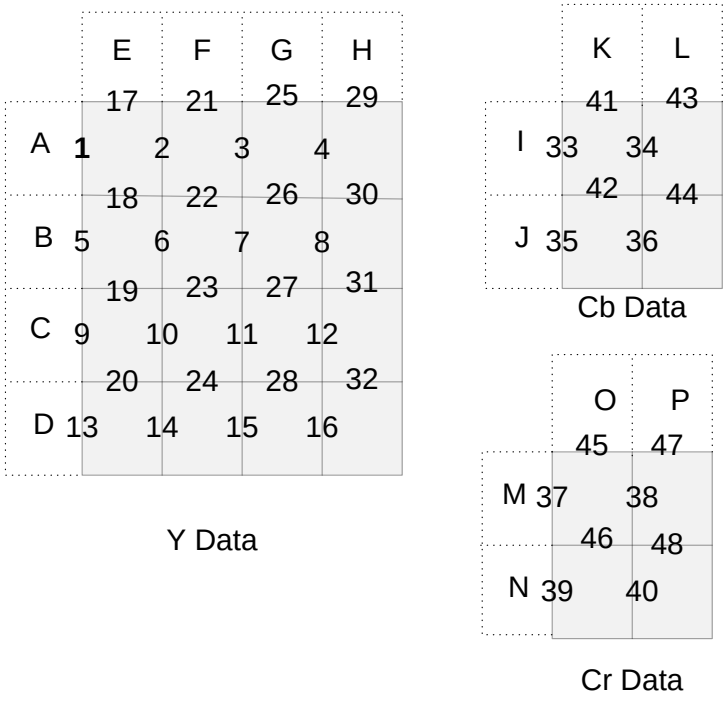

Figure 13: Filter Order

The restriction imposed by H.264 standard is that horizontal filtering should precede vertical filtering. The filtering order in Figure 13 complies with the previous restriction and it was chosen to serve the need of our scalable architecture and to provide better data reuse.

3.2 Top Level Architecture

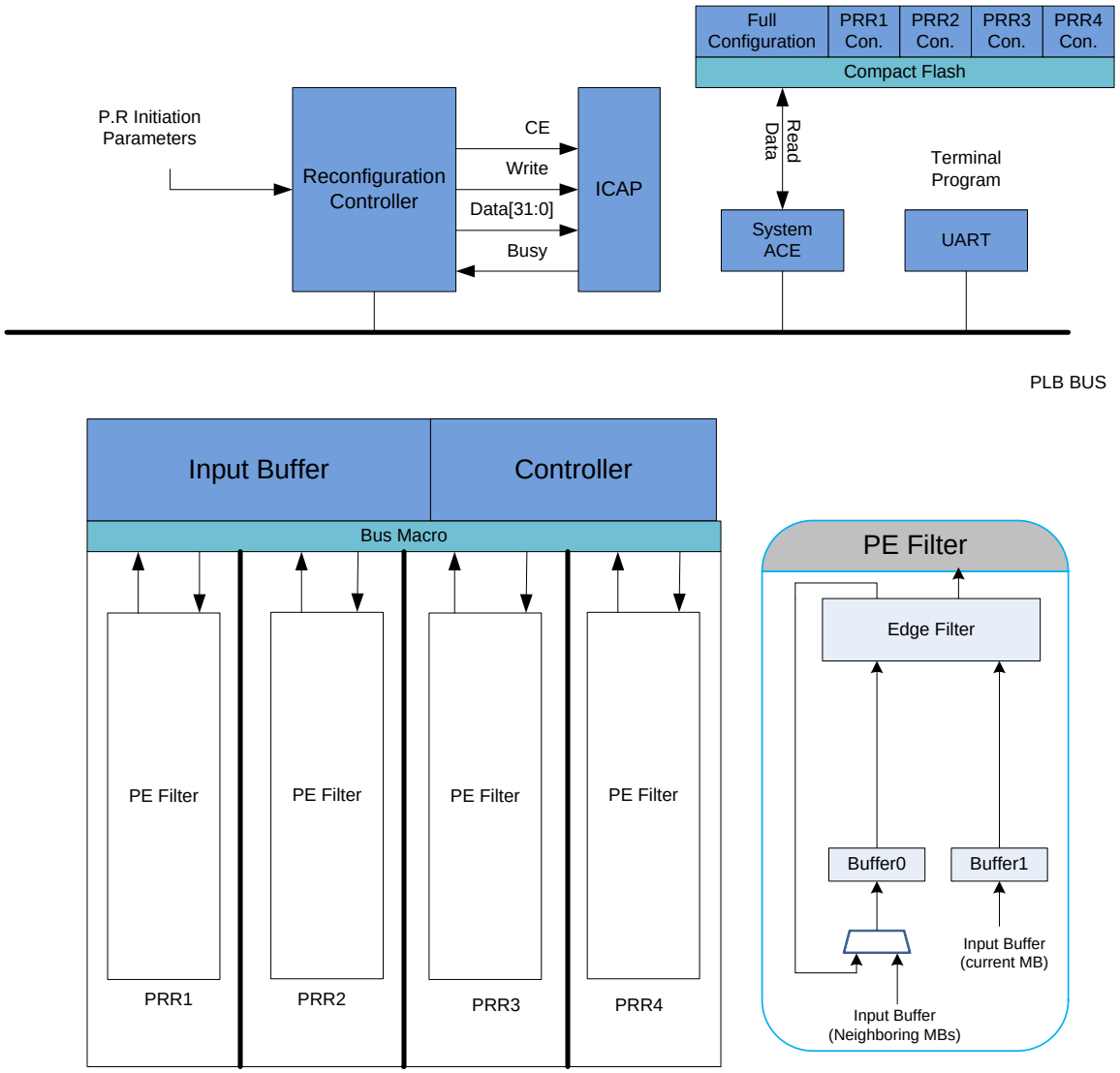

Figure 14: Top Level Architecture of Partially reconfigurable DBF

The top level architecture of scalable DBF is shown in Figure 14. All the datapaths are 32-bit wide. The architecture consists of static region and reconfigurable regions. The static region includes the input buffer, controller, and reconfiguration controller. The 80x32 bit input buffer is

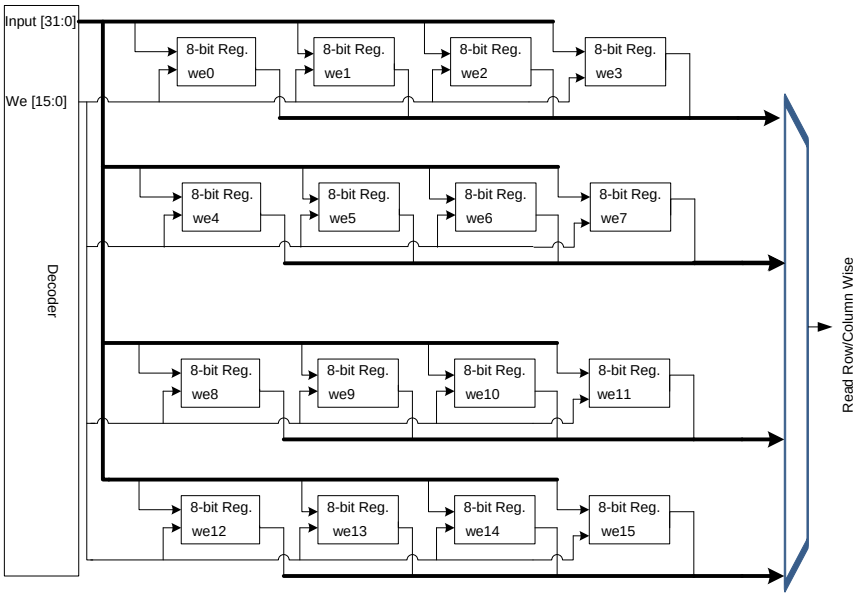

Figure 15: Basic Building Block of the Input Buffer

used to store the pixels in a 16x16 MB until all the edges in the MB are fully filtered. The buffer is four dual port memory that is made from registers. As shown in Figure 15, the basic building block of the input buffer has a 4x4 pixel array structure, where each pixel is stored in 8-bit register. The controller assigns the write enable signal to select the specific row/column for data writing, and generates address signals for data fetching. The proposed DBF architecture can include up to four Partial Reconfiguration Regions (PRRs), i.e., from PRR1 to PRR4. Each of PRRs holds the modular DBF engine. Each PRR can perform filtering of each edge in a 4x4 block. For example, if 4 PRRs are used, then four blocks can be filtered concurrently on four distinct rows (columns) in the case of horizontal (vertical) filtering, respectively, as shown in Figure 16. The reconfiguration is done using Internal Configuration Access Port (ICAP) which allows for internal device reconfiguration during run-time. In this paper, Microblaze processor is used as reconfiguration controller to provide management for reconfiguration and also allow the

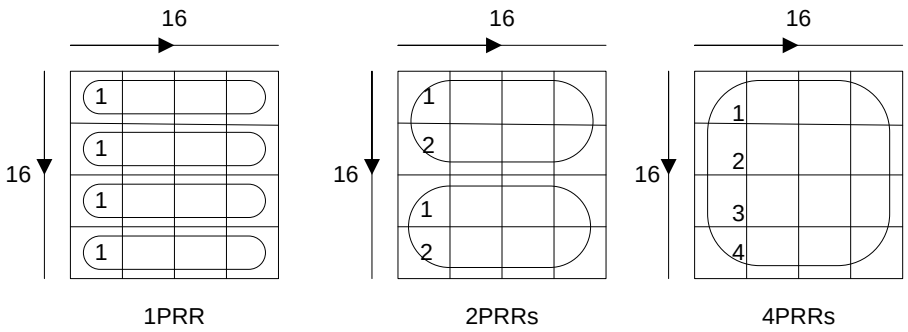

Figure 16: Degree of parallelism, depending on the number of active PRRs

user to trigger reconfiguration through UART. The controller also provides the necessary signals for reconfiguration to be done using ICAP. “CE” signal is the chip enable for ICAP interface and “Write” is write enable signal. “Data” is the handshaking signal to indicate that ICAP is busy and cannot take new data for reconfiguration. System ACE is used as an interface for compact flash card. UART is used as a user interface through HyperTerminal. The Bus Macro (BM) is used as an interface for the signals connecting the static and the reconfigurable regions.

3.2 Partial Reconfigurable Module

Figure 14 shows the internal structure of a PE filter. This module consists of two 4x4 pixel buffers and an edge filter. There is an initial latency of 3 clock cycles to fetch the data from the input buffer. The data after first filtering operation are sent to buffer0 for filtering the next edge while new data from the neighboring 4x4 block are stored at buffer1. After the data are filtered second time, they are stored back to the input buffer as they are needed later for vertical filtering which uses the same flow as horizontal filtering. Because of the versatile structure of the input buffer, we can read and write data column-wise or row-wise on the fly. It takes 4 clock cycles to filter one edge. If two or four PRRs become active, then, two or four distinct 4x4 blocks can be concurrently filtered, respectively.

3.3 Experimental Results

The scalable architecture is implemented using Xilinx Virtex-4 (XC4VFX60-FF1152) ML410 board. Synthesis is done using Xilinx ISE Foundation 9.2i. Figure 17 shows the PAR map of the reconfigurable architecture. The full and partial bitstreams are generated through Xilinx PlanAhead 10.1 Tool [20]. From partial reconfiguration point of view, the Virtex-4 architecture has finer reconfiguration granularity as the configuration frame resolution is 16 CLBs in height. In Virtex-II and Virtex-II pro, the configuration frame was for the whole CLB column [5]. Therefore, we can have multiple reconfiguration regions at any given column and the reconfiguration can be performed faster compared to previous family of Xilinx FPGA devices. Among different configuration modes supported in Xilinx FPGAs, ICAP [21] is used for dynamic partial reconfiguration in our modular design. The partial bitstreams are stored on Compact Flash card and the reconfiguration is triggered by the software running on Microblaze processor which provides the necessary handshaking signals to the ICAP interface (see Table 4).

Table 4: ICAP Interface Ports

Port Name	Direction	Description
CLK	INPUT	ICAP INTERFACE CLOCK
CE	INPUT	CONFIGURATION ENABLE
WRITE	INPUT	WRITE/READ SIGNAL
I[31:0]	INPUT	ICAP WRITE DATA BUS
O[31:0]	OUTPUT	ICAP READ DATA BUS
BUSY	OUTPUT	BUSY(BUFFER IS FULL)

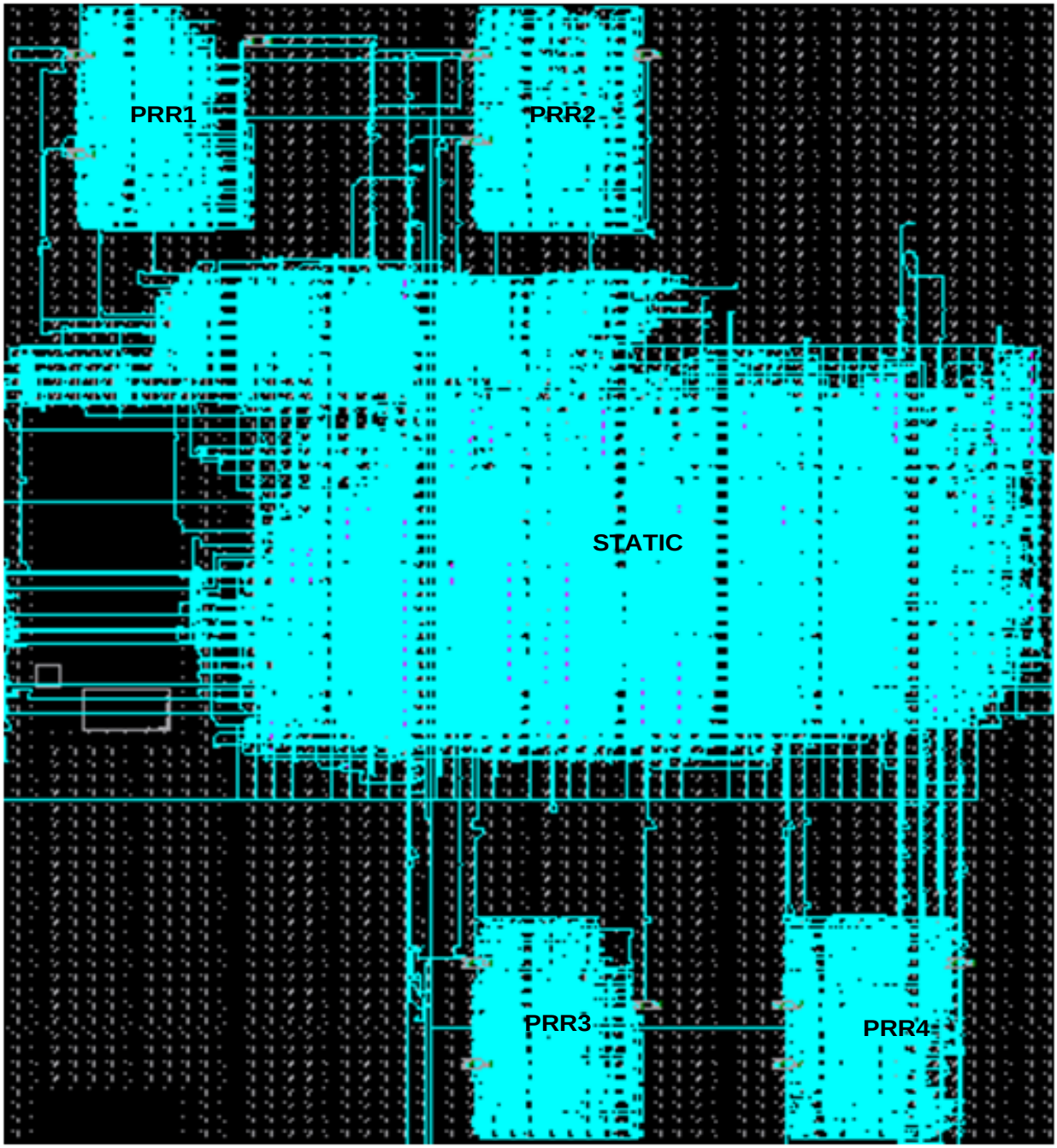

Figure 17: PAR map of reconfigurable architecture

Table 5 shows the performance comparison with previous work and the scalability factor when using multiple configuration regions to enhance performance and parallelism. It is shown that the filtering cycles are reduced significantly when more number of PEs are used to achieve speed-up.

Table 6 shows the time required for filtering operation for each frame at a clock frequency of 50 MHz. It shows that different reconfiguration modes can be selected to increase its throughput, depending on the input resolution of video sequences. In addition, the proposed architecture can be self-reconfigured during run-time through dynamic partial reconfiguration to adapt to the changing frame rate. Therefore, reduction of power consumption and hardware resources can be achieved while the proposed reconfigurable architecture for DBF algorithm maintains its required performance.

Table 7 shows the synthesis results of the static and reconfigurable regions, and their device utilization. Static region is always active after initial reconfiguration, and used for self-reconfiguration control, data buffering, and prediction of required computational complexity. Reconfigurable regions are used to support scalable DBF architecture so that varying computational loads during run-time can be met with higher flexibility.

Table 8 shows the partial bitstream size and the configuration time. There is a difference between Virtex-II ICAP and Virtex-4 ICAP. The former has 8-bit data port operating at 50 MHz, while the later has 32-bit data port operating at 100 MHz. This implies a reconfiguration rate of 3.2 Gbps for the Virtex 4 device family [22].

Table 5: Design Comparisons, N=1,2 OR 4

Architecture	Filtering Cycles/MB	Memory Size	Filters
[23]	250	160x32	1
[24]	300	16x32	1
[25]	204	2x96x32	1
[15]	192	160x32	1
Proposed work	$\frac{192}{N}$	(80+Nx8)x32	N

Table 6: Various Reconfigurable Modes to Support Different Resolutions of Video Sequences

Display Type	Type of Reconfigurable architecture	Estimated Filter Processing time for each frame @ 50MHz
VGA (640×480)	1PRR	4.68ms
	2PRR	2.37ms
720×480	1PRR	5.26ms
	2PRR	2.67ms
	4PRR	1.37ms
1920×1080	1PRR	32ms
	2PRR	15.85ms
	4PRR	8.2ms
2560×1920	1PRR	75ms
	2PRR	38ms
	4PRR	20ms

Table 7: Hardware Resources

Module	LUTs (Device Utilization)	Slice Flip Flops (Device Utilization)	BRAMs (Device Utilization)
Reconfigurable (4 PRRs)	2784(5.5%)	0(0%)	8(3.4%)
Static Region (Controller, Input Buffer, ICAP Controller)	14738(29.1%)	5871(11.61%)	32(13.79%)

Table 8: Bitstream Information

Bitstreams	Size per bitstream (Bytes)	Configuration time per bitstream (µs)
PRR1,2,3, and 4	39032	97.58

3.4 Summary

We propose self-reconfigurable architecture for a scalable H.264/AVC DBF using FPGA dynamic partial reconfiguration. The scalable architecture can perform filtering up to four distinct blocks at the same time, reducing filtering clock cycles significantly and improving its throughput. Our DBF engine has the ability to adapt itself to diverse application needs which can be used to support different resolutions and frame rates dynamically. The number of processing elements can be changed during run-time using dynamic partial reconfiguration through ICAP controller.

CHAPTER FOUR: BIT-RATE AWARE DEBLOCKING FILTER

4.1 Complexity Reduction for Deblocking Filter

What determines an edge needs to be filtered or not is the BS value. BS=0 means no filtering is done across the edge, this is due to edges of blocks have zero residual data or they are copied directly from a previous frame without a difference in motion vectors. The complexity of the filtering process varies with video characteristics and bit-rate [26]. This variation comes into effect by the percentage of edges that may not be filtered due to BS=0. Our experiments show that when lowering the bit-rate of video sequences, the percentage of edges that need not to be filtered increase. This is because of more residual data are quantized to zero, and larger block sizes are used for motion compensation, so BS=0. Therefore, the computational complexity can be reduced by the increased presence of skipped MBs, thus reducing the processing cycles needed for filtering operation.

4.2 Simulation Results and Evaluation

Figure 18-Figure 25 shows four QCIF video sequences with a video format of IPPPPPP..., and using the JM15.0 software [27] (with coding parameters set as follows: one reference frames, motion estimation search range of ±16, 100 frames), we can see the distribution of BS values and block size mode for different bit-rates. And while the bit-rate decreases the number of edges that need not to be filtered (BS=0) increases. In our experiments we found out that those edges are gathered in some MBs, and thus those MBs could be skipped. Thereby decreasing the computational time needed for filtering operation.

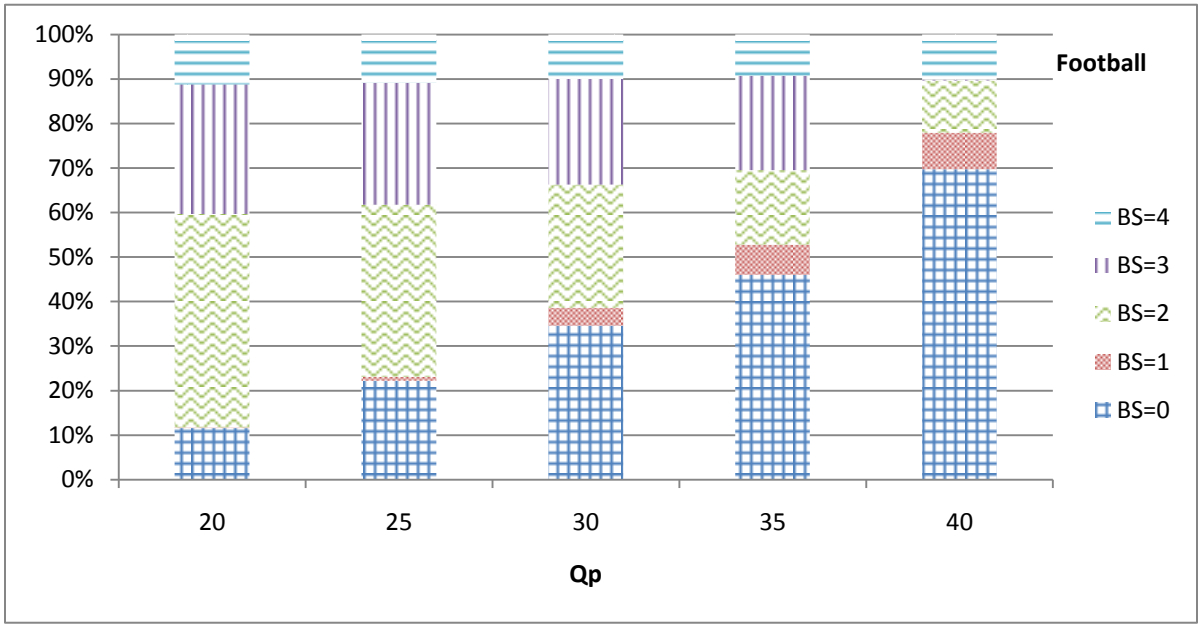

Figure 18: BS Distribution for Football

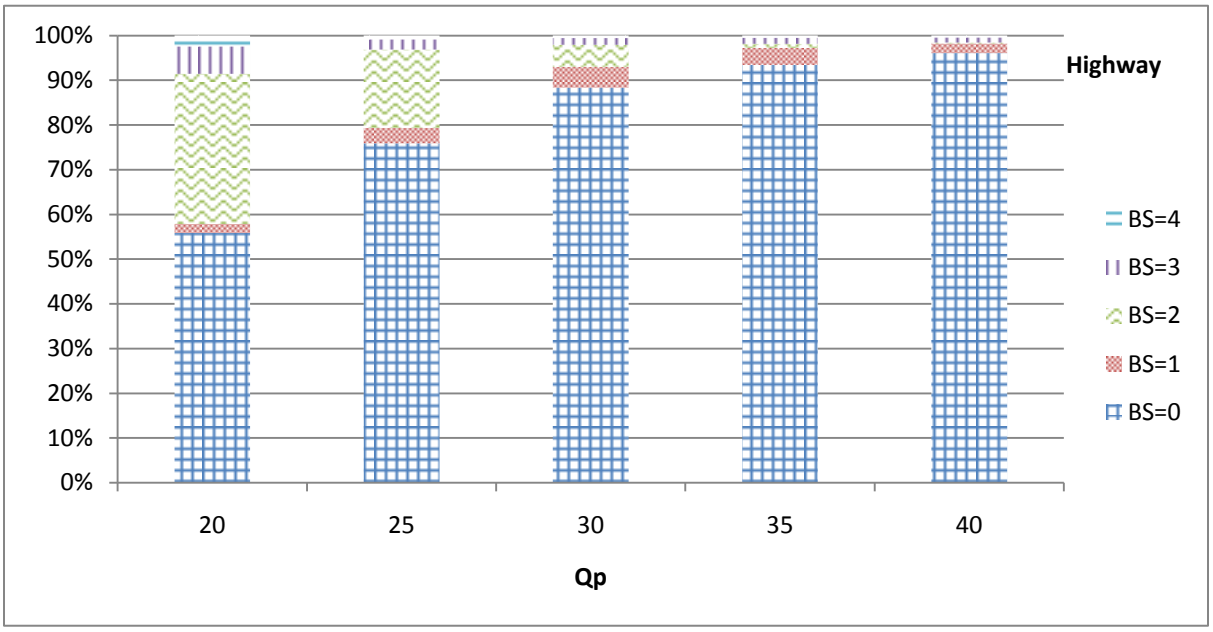

Figure 19: BS Distribution for Highway

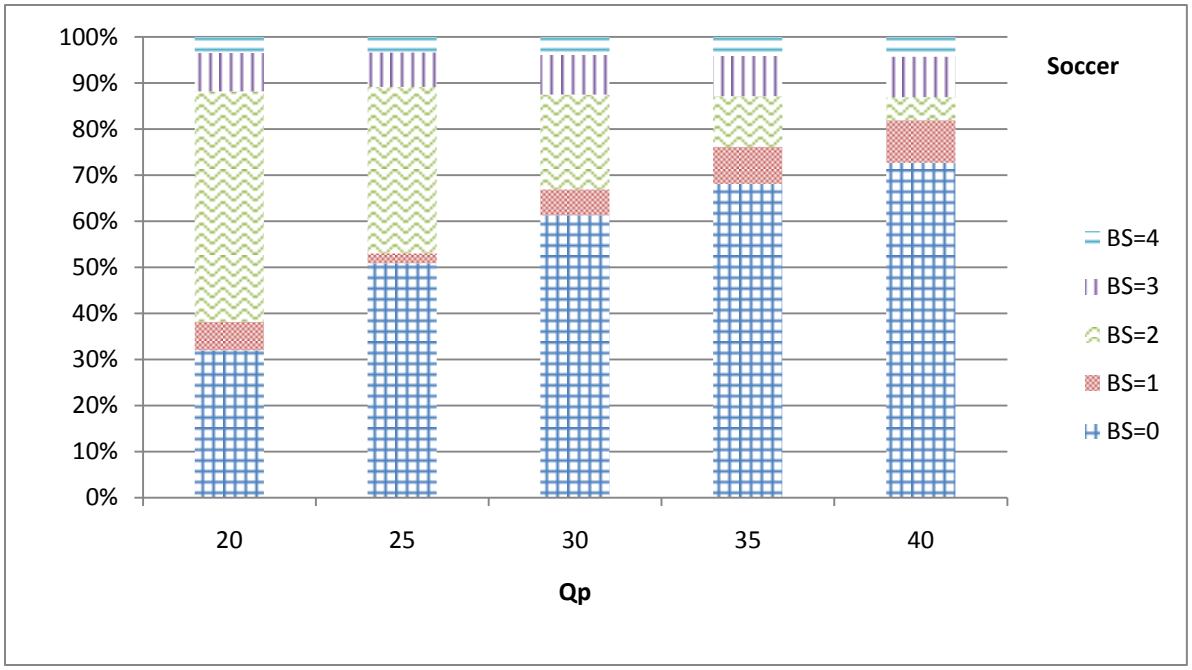

Figure 20: BS Distribution for Soccer

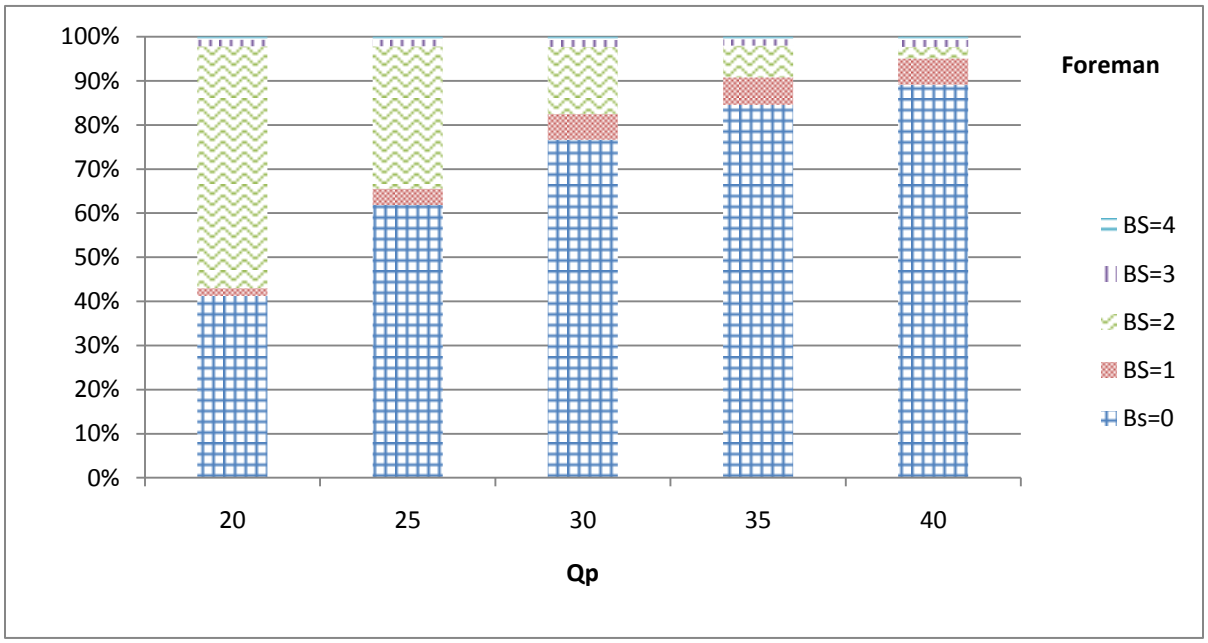

Figure 21: BS Distribution for Foreman

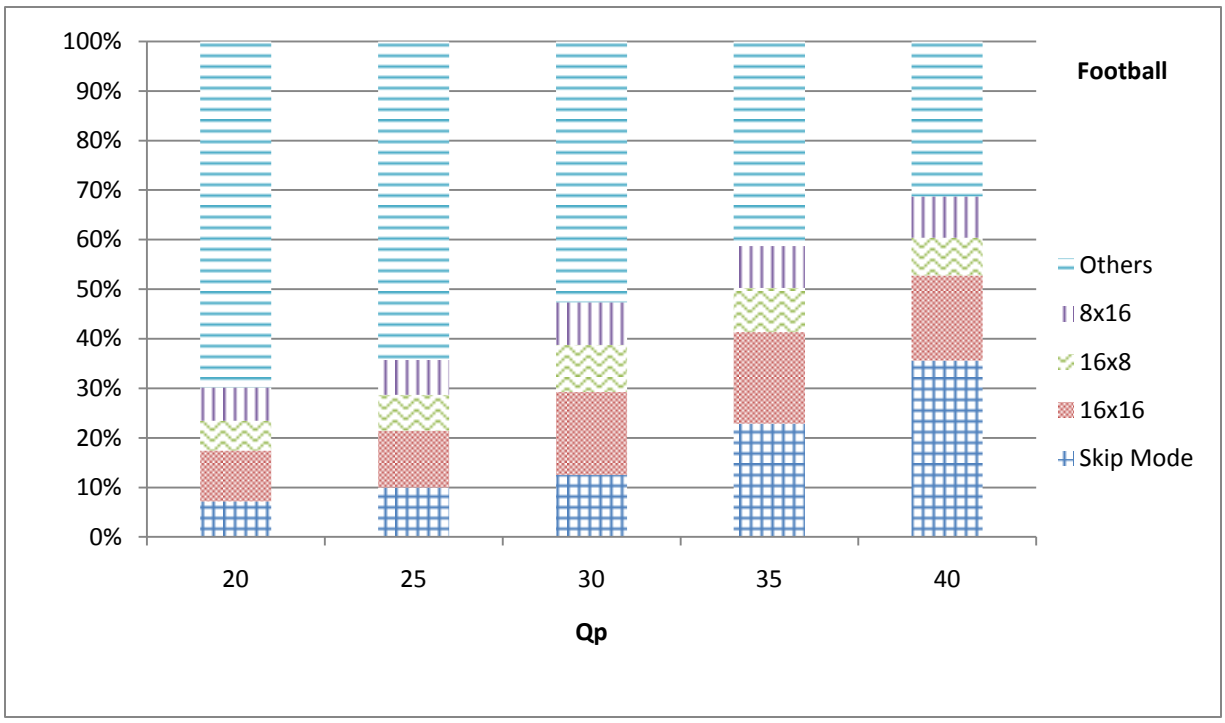

Figure 22: Block Size Mode Distribution for Football

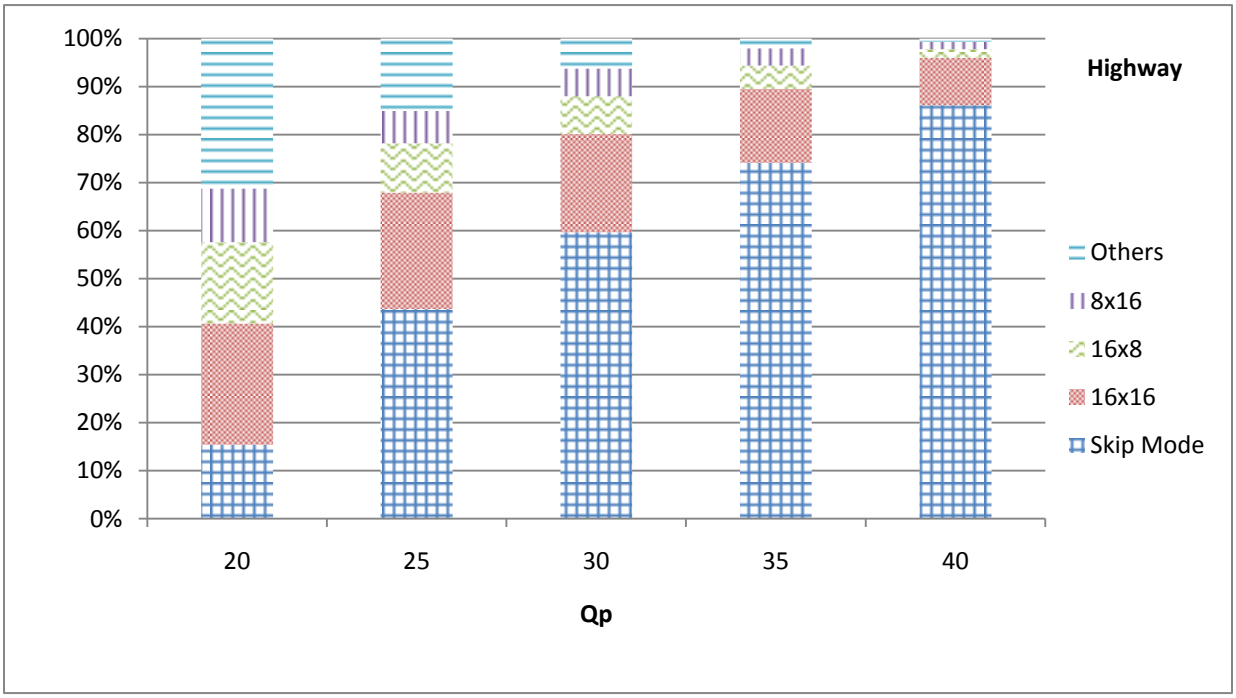

Figure 23: Block Size Mode Distribution for Highway

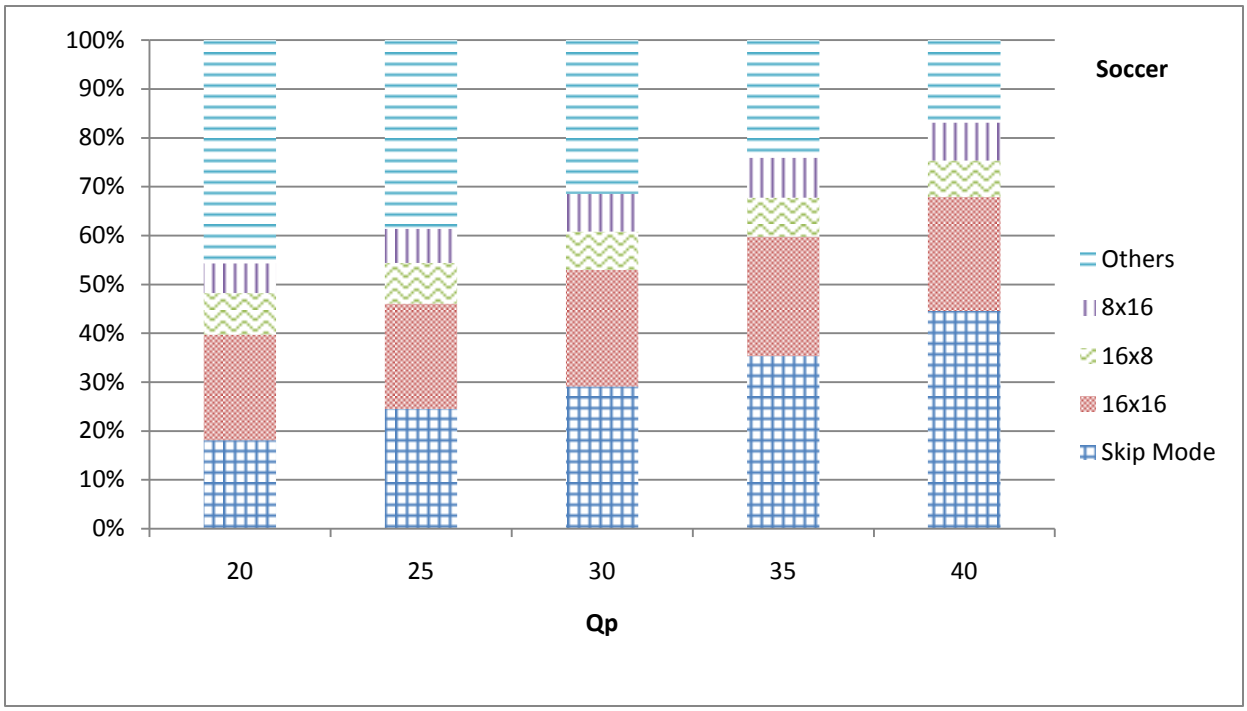

Figure 24: Block Size Mode Distribution for Soccer

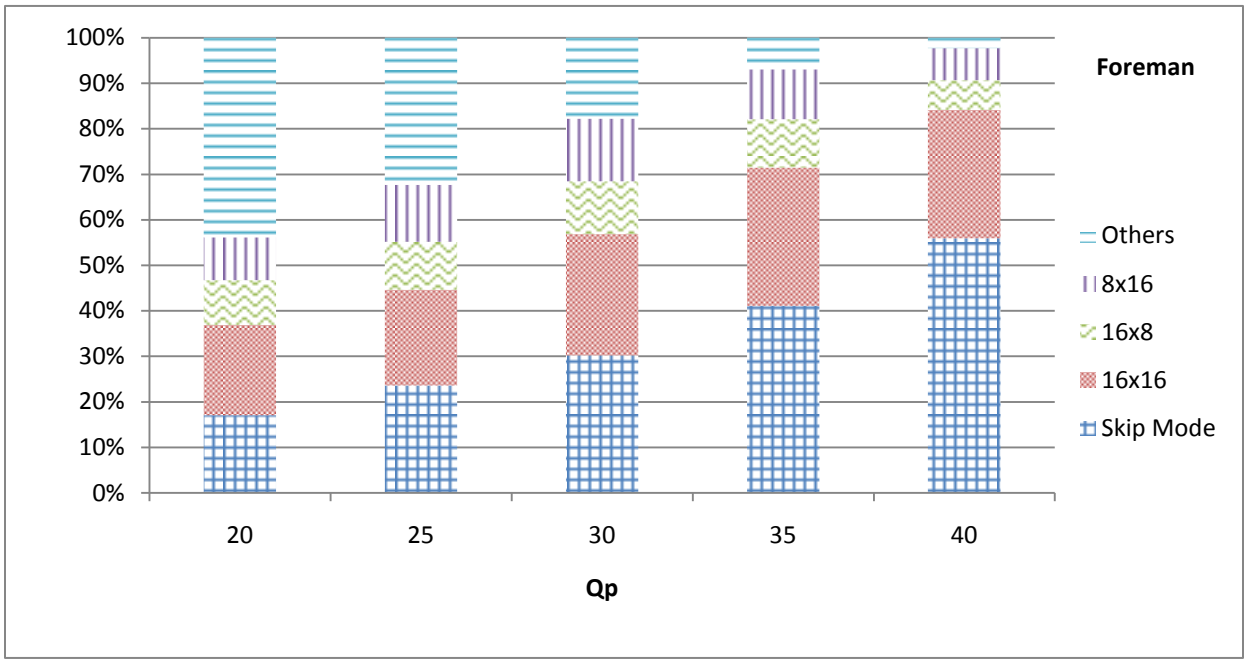

Figure 25: Block Size Mode Distribution for Foreman

4.3 Scalable Architecture Adaptability to Compression Ratio and Motion Activity

Table 9: Skipped MacroBlocks at Different Quantization Parameters

Sequence	Foreman	Container	Football	Highway	Soccer
Qp=20	30.8%	69.51%	8.88%	51.2%	26.22%
25	55.87%	87.21%	17.33%	73.63%	44.22%
30	78.05%	96.38%	33.10%	92.32%	59.79%
35	89.76%	98.49%	48.41%	97.58%	71.64%
40	95.53%	98.91%	61.3%	98.38%	81.08%

Table 9 clearly shows at different Quantization parameters (Qp) considerable reduction in computational complexity. For low/moderate motion video sequences, the PRRs can be reduced to half while maintaining the performance at Qp=25. For high motion video sequences, e.g., Soccer and Football, the PRRs can be reduced by half at Qp=35. In this paper, the absolute sum of Motion Vectors (MVs) is used to differentiate between low/moderate and high motion video sequences. Table 10 shows the absolute sum of motion vectors for different video sequences. We use a threshold, i.e., TH_{MV}=55000, obtained from our various simulation results. If the MV sum is greater than the threshold, then the video sequence is considered to have high motion activity. Figure 26 gives more details on the adaptability of the number of PEs depending on motion activity and various bit-rates.

Table 10: Motion Vector Sum for Various Video Sequences (10 FRAMES USED AT QP=30)

QCIF Sequence	MV sum
football	190583
soccer	125983
foreman	53935
Highway	31446
news	7753
clair	5144
contrainer	574

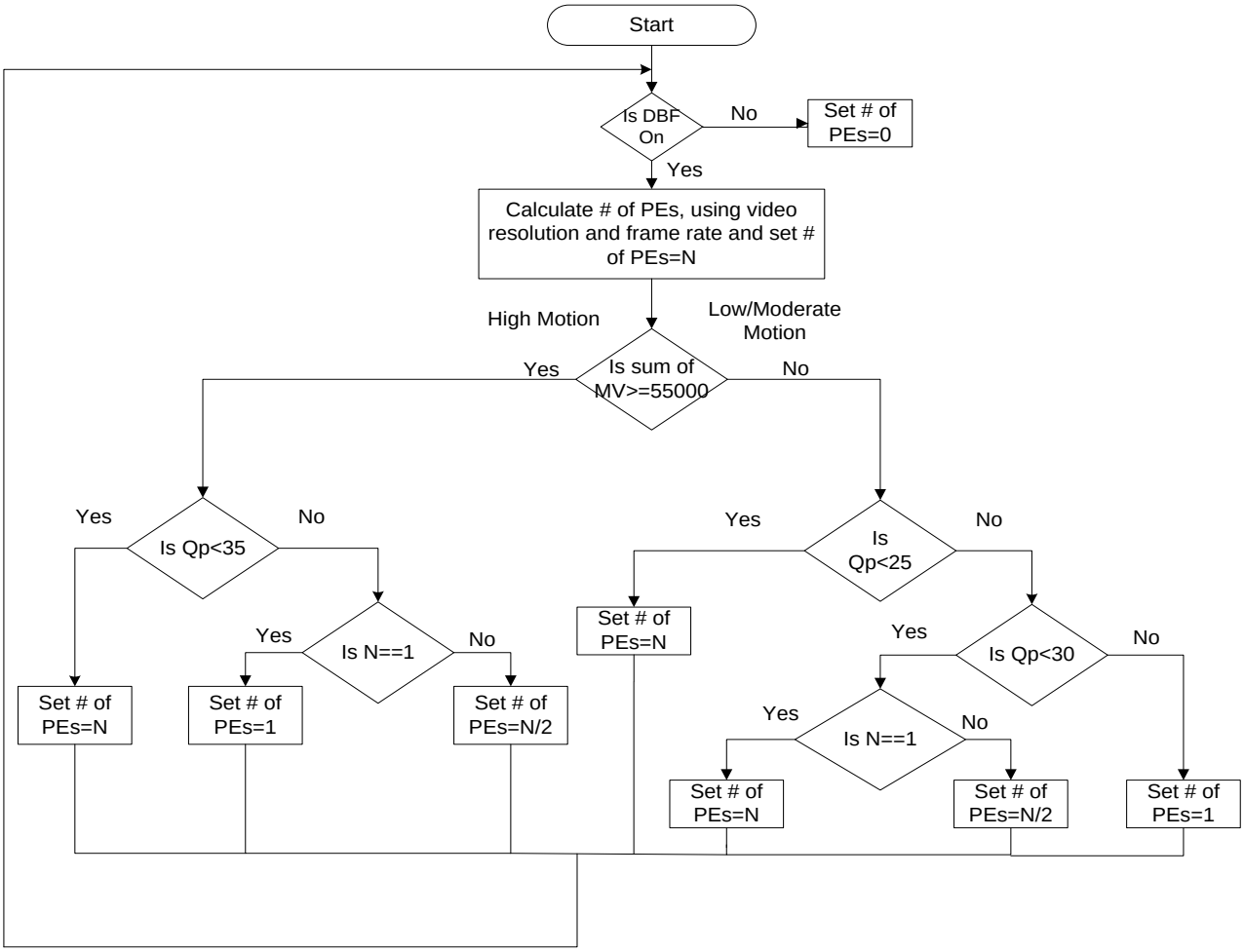

Figure 26: PEs adaptability to changing bit-rate and motion activity

4.4 Summary

We propose self-reconfigurable architecture for a scalable H.264/AVC DBF using FPGA dynamic partial reconfiguration. We combine the algorithmic and hardware scalability to synergize the performance. The scalable architecture can perform filtering up to four distinct blocks at the same time, reducing filtering clock cycles significantly and improving its throughput. Moreover, at lower bit-rates, computational cost is reduced greatly by the presence of skipped MBs, which in turn improves throughput and less hardware needed for filtering operations. Our DBF engine has the ability to adapt itself to diverse application needs which can be used to support various resolutions, frame rates, and bit-rates dynamically by reconfiguring processing elements during run-time.

REFERENCES

[1] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of JointVideo Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.

[2] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Tranactions. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560–576, July 2003.

[3] I. E. G. Richardson. H.264 and MPEG-4 Video Compression – Video Coding for Next-generation Multimedia. John Wiley & Sons Ltd, 2003.

[4] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, and Marta Karczewicz, "Adaptive Deblocking Filter", *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 614-619, July 2003.

[5] LYSAGHT, P., BLODGET, B., MASON, J., YOUNG, J.A.Y.J. and BRIDGFORD, B.A.B.B. 2006. Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2006, 1-6.

[6] XILINX 2006. Early Access Partial Reconfiguration. *User Guide 208*.

[7] XILINX 2007a. Difference-Based Partial Reconfiguration. *Application Note 290*.

[8] C. Arbelo, A. Kanstein, S. Lopez, J.F. Lopez, M. Berekovic, R. Sarmiento, and J.-Y. Mignolet, "Mapping Control-Intensive Video Kernels onto a Coarse-Grain Reconfigurable Architecture: the H.264/AVC Deblocking Filter", *Design, Automation & Test in Europe Conference & Exhibition*, pp.1-6, 2007.

[9] Warrington, S., Shojania, H., and Sudharsanan, S., "Performance Improvement of the H.264/AVC Deblocking Filter Using SIMD Instructions", *IEEE Proc. ISCAS*, pp.2697-2700, 2006.

[10] C. C. Cheng and T. S. Chang, "An Hardware Efficient Deblocking Filter for H.264/AVC", *IEEE International Conference on Consumer Electronics*, pp.235- 236, Jan. 2005.

[11] G. Khurana, A. A. Kassim, T. P. Chua, and M. B. Mi., " A pipelined hardware implementation of In-loop Deblocking Filter in H.264/AVC." *IEEE Transactions on Consumer Electronics*, pp.536 – 540, 2006.

[12] B. Sheng, W. Gao, and D. Wu. "An Implemented Architecture of Deblocking Filter for H.264/AVC." *Proceedings - International Conference on Image Processing, ICIP*, pp.665 – 668, 2004.

[13] Hao, W.N. and Radetzki, M., "A Data Traffic Efficient H.264 Deblocking IP", *IEEE Int. Symp. on Circuits and Systems*, pp.3430-3433, 2008.

[14] Min, K.Y. and Chong, J.W., " A Memory and Performance Optimized Architecture of Deblocking Filter in H.264/AVC*", IEEE Conf. on Multimedia and Ubiquitous Engineering*, pp.220-225, 2007.

[15] Lingfeng Li, Satoshi Goto, and Takeshi Ikenaga, " A Highly Parallel Architecture for Deblocking Filter in H.264/AVC", *IEICE Transactions on Information and Systems*, Vol.E88-D, no.7, pp.1623-1629, July 2005.

[16] L. Li, S. Goto, and T. Ikenaga. An efficient deblocking filter architecture with 2-dimensional parallel memory for H.264/AVC. In ASP-DAC '05: *Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 623–626, 2005.

[17] S. Shih, C. Chang, and Y. Lin. A near optimal deblocking filter for H.264 advanced video coding. *Proceedings of the Asia and South Pacific Design Automation Conference*, pp.170 – 175, 2006.

[18] Mustafa Parlak and Ilker Hamzaoglu, "Low Power H.264 Deblocking Filter Hardware Implementations", *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, May 2008.

[19] Yoshinori Hayashi, Tian Song, Eiji Koeta, and Takashi Shimamoto, "Fast Deblocking Filter Implementation Method and It's Architecture for H.264/AVC", *International Conference in Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI-CON2008)*, pp.I-465-468, May 2008.

[20] Xilinx Inc., PlanAhead 10.1 User guide, http://www.xilinx.com/support/documentation/sw_manuals/PlanAhead_UserGuide.pdf.

[21] XILINX, Virtx-4 FPGA Configuration User Guide, June 2009.

[22] XILINX, Virtex-4 FPGA User Guide, August 2007.

[23] T. M. Liu, W. P. Lee, T. A. Lin, and Chen-Yi Lee, "A memory efficient deblocking filter for H.264/AVC video coding*", in Proceedings of IEEE International Conference Symposium on Circuit and Systems,* vol. 3, pp. 2140-2143, May 2005.

[24] C.-C. Cheng, T.-S. Chang, and K.-B. Lee, "An in-place architecture for deblocking filter in H.264/AVC", *IEEE Transactions on Circuits and Systems II*, vol. 53, no. 7, pp. 530-534, July 2006.

[25] Ke Xu and Chiu-Sing Choy , "A Five-Stage Pipeline, 204 Cycles/MB, Single-Port SRAM-Based Deblocking Filter for H.264/AVC", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 3, pp. 363-374, March 2008.

[26] Ke Xu, T. M. Liu, J. I. Guo, and C. S. Choy, "Methods for Power/throughput/area Optimization of H.264/AVC Decoding", *Journal of Signal Processing Systems*" DOI 10.1007/s11265-009-0408-6, October 2009.

[27] H.264/AVC Reference Software Version JM15. Available from http://iphome.hhi.de/suehring/tml/.